patrones.

https://coderwall.com/team/47-degrees

anti.patrones

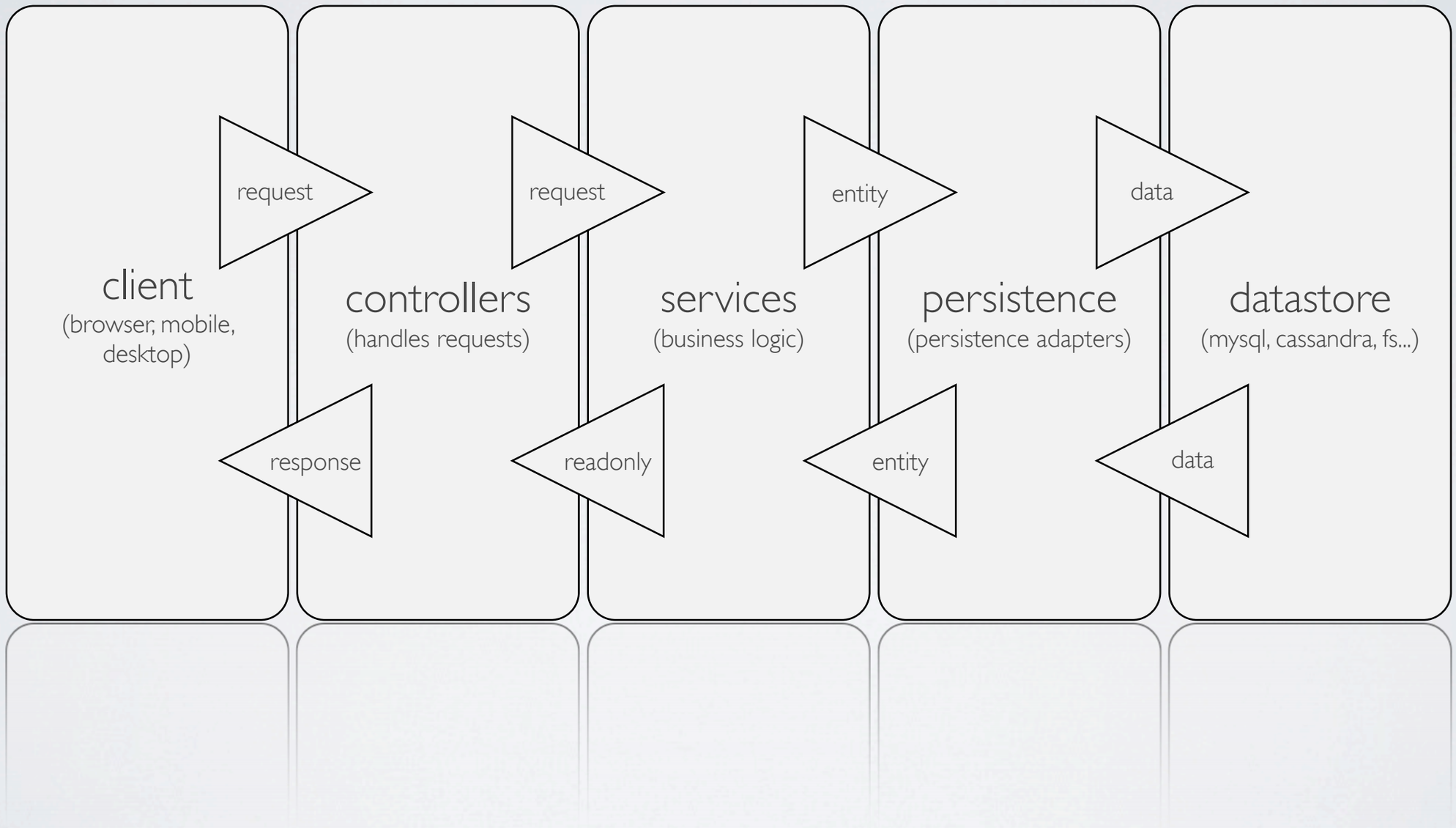
anti.patrones

arquitectura, diseño y dialectos del software

patrones

arquitectura

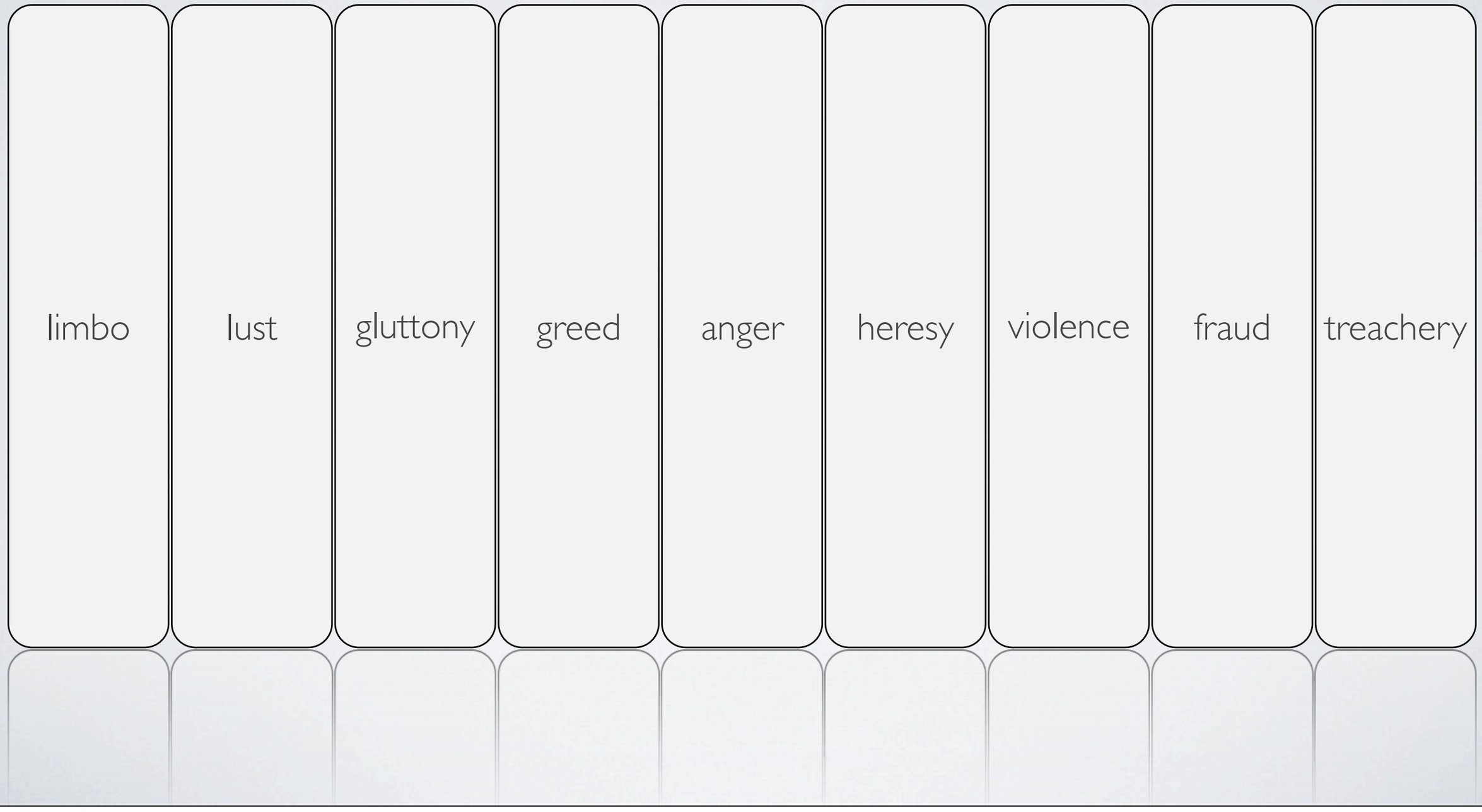
layers



anti.patrones

arquitectura

layers / YAFL (yet another fucking layer)



anti.patrones

arquitectura

layers / YAFL (yet another fucking layer)

*Dante's inferno layers

patrones

arquitectura

mvc

patrones
arquitectura

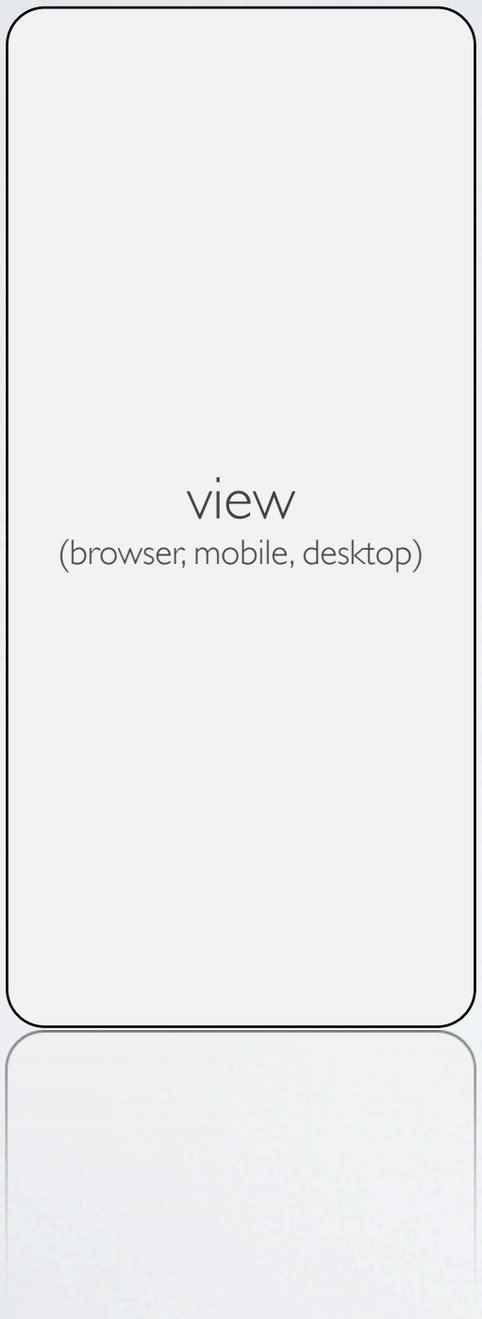
mvc

*Great for applications

patrones
arquitectura

mvc

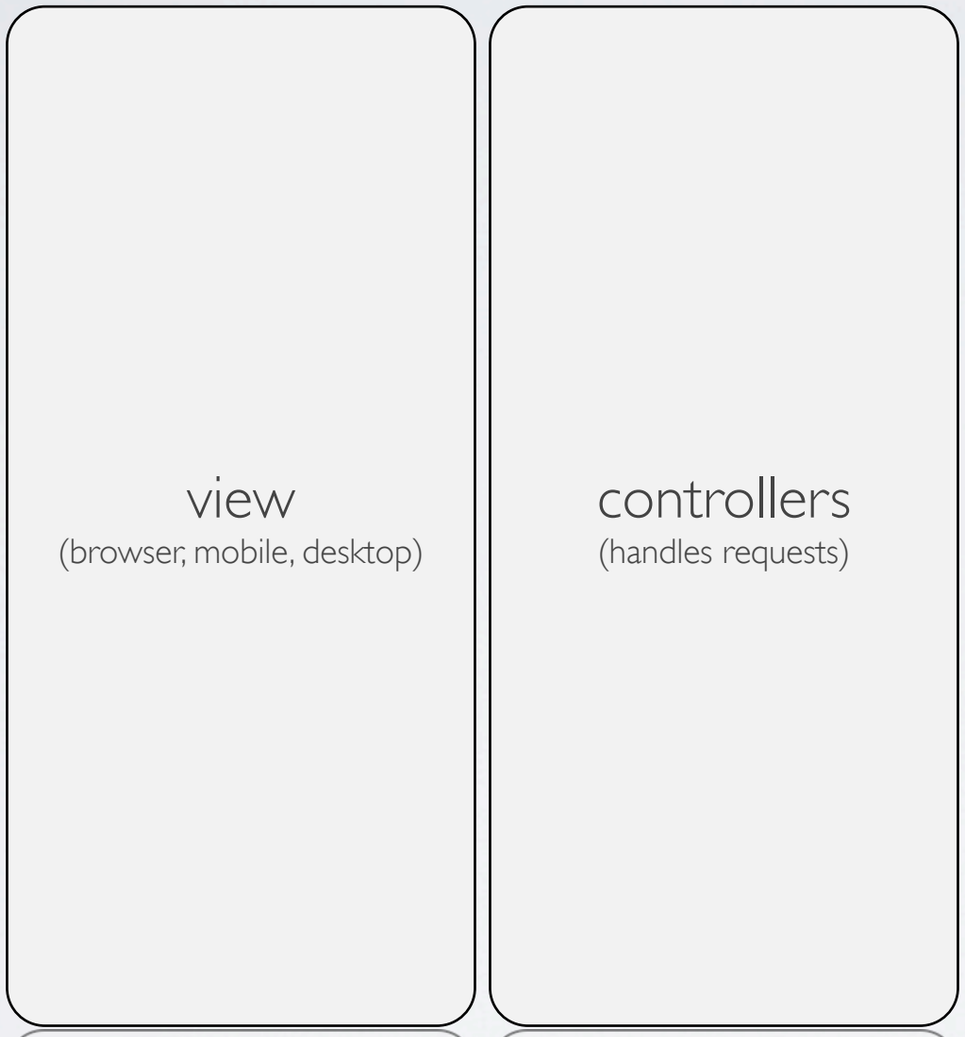
*Great for applications



patrones
arquitectura

mvc

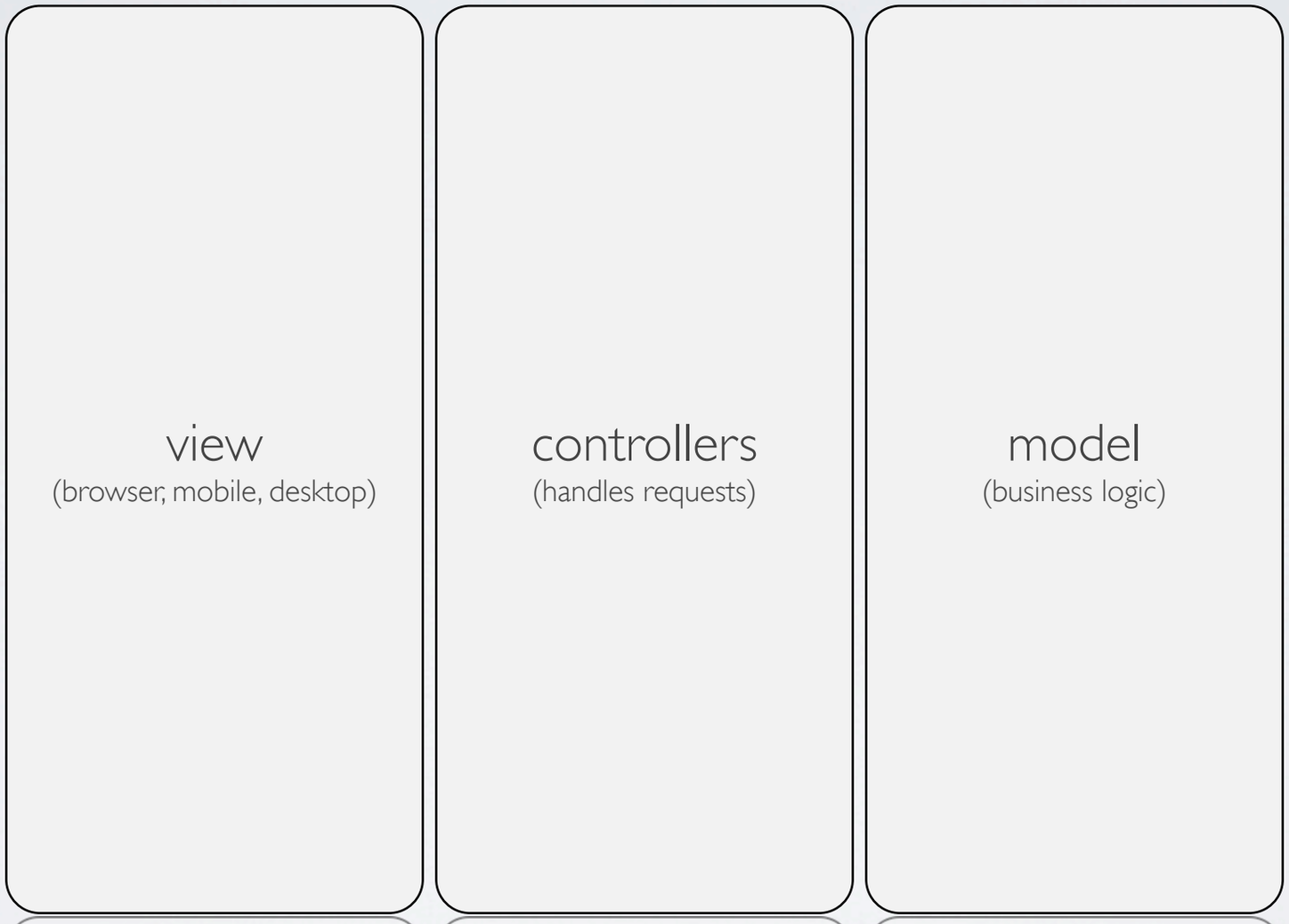
*Great for applications



patrones
arquitectura

mvc

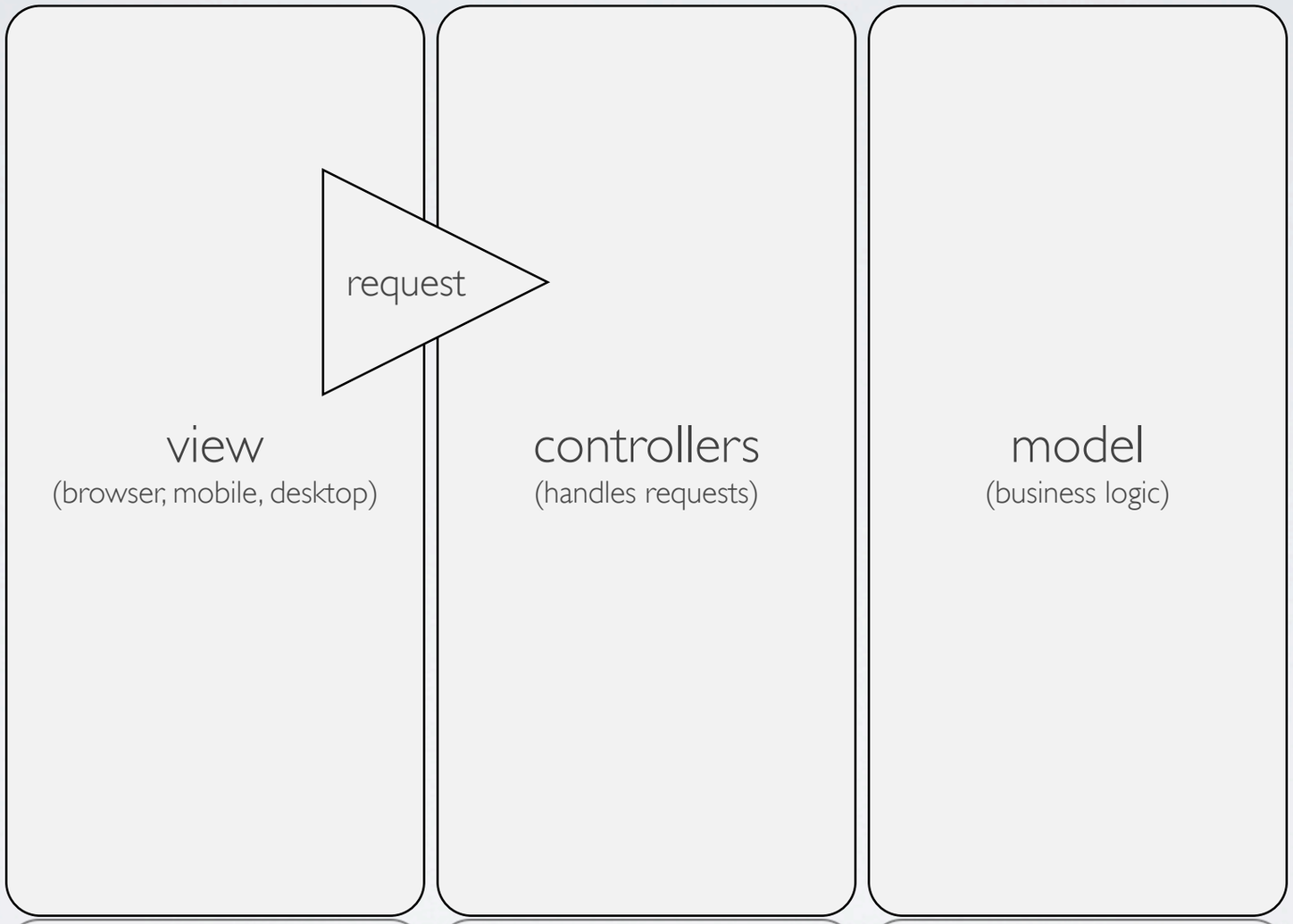
*Great for applications



patrones
arquitectura

mvc

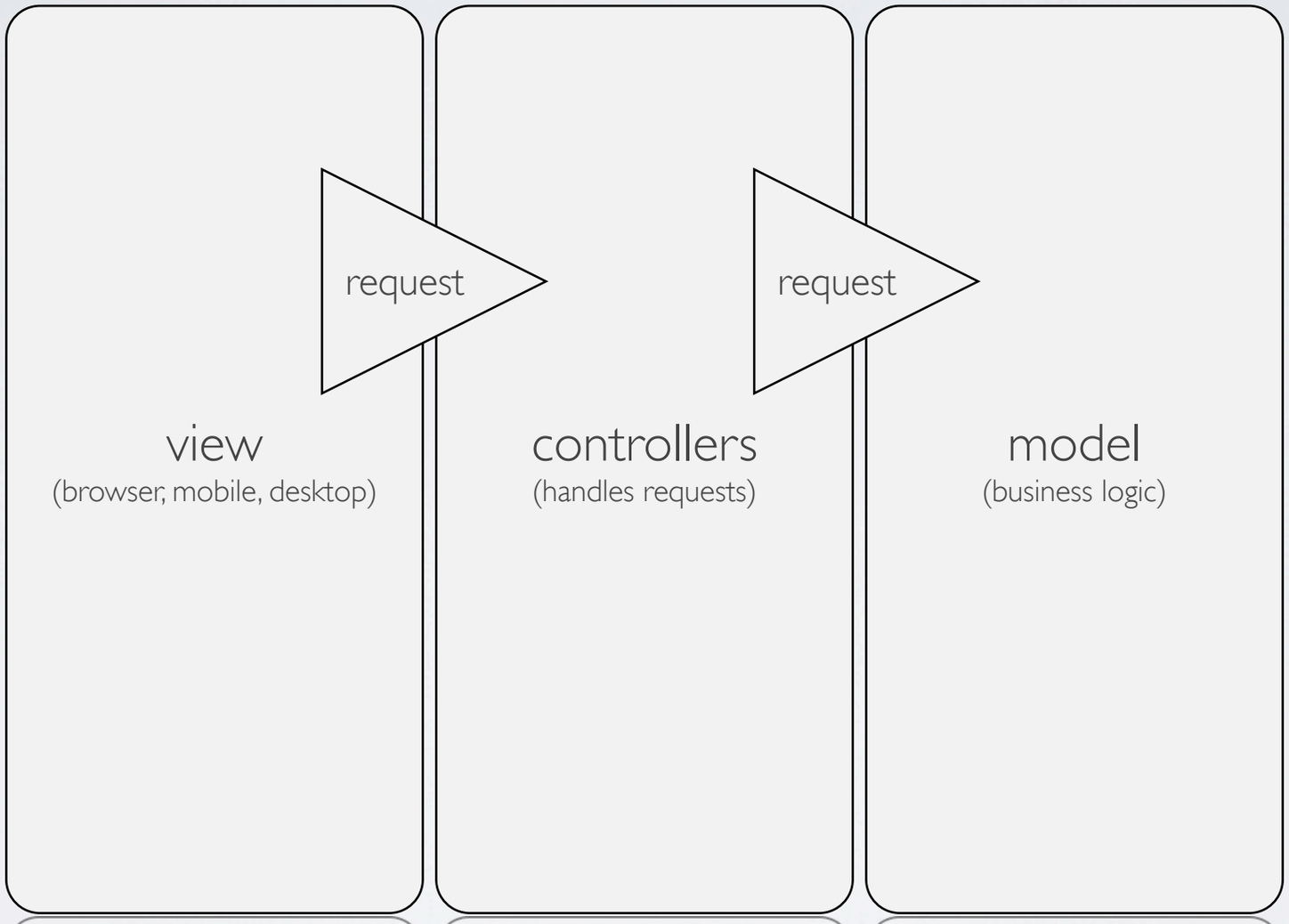
*Great for applications



patrones
arquitectura

mvc

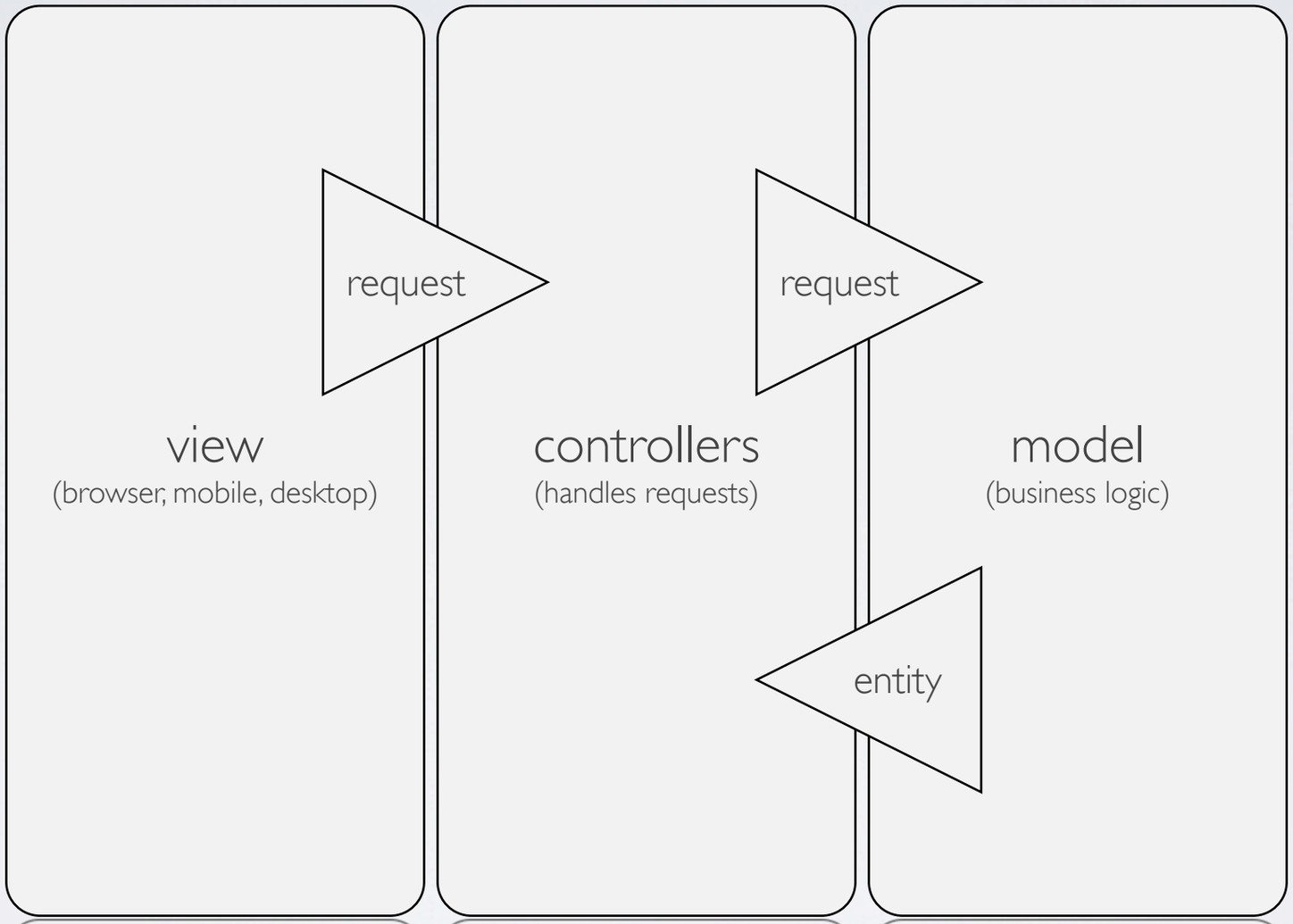
*Great for applications



patrones
arquitectura

mvc

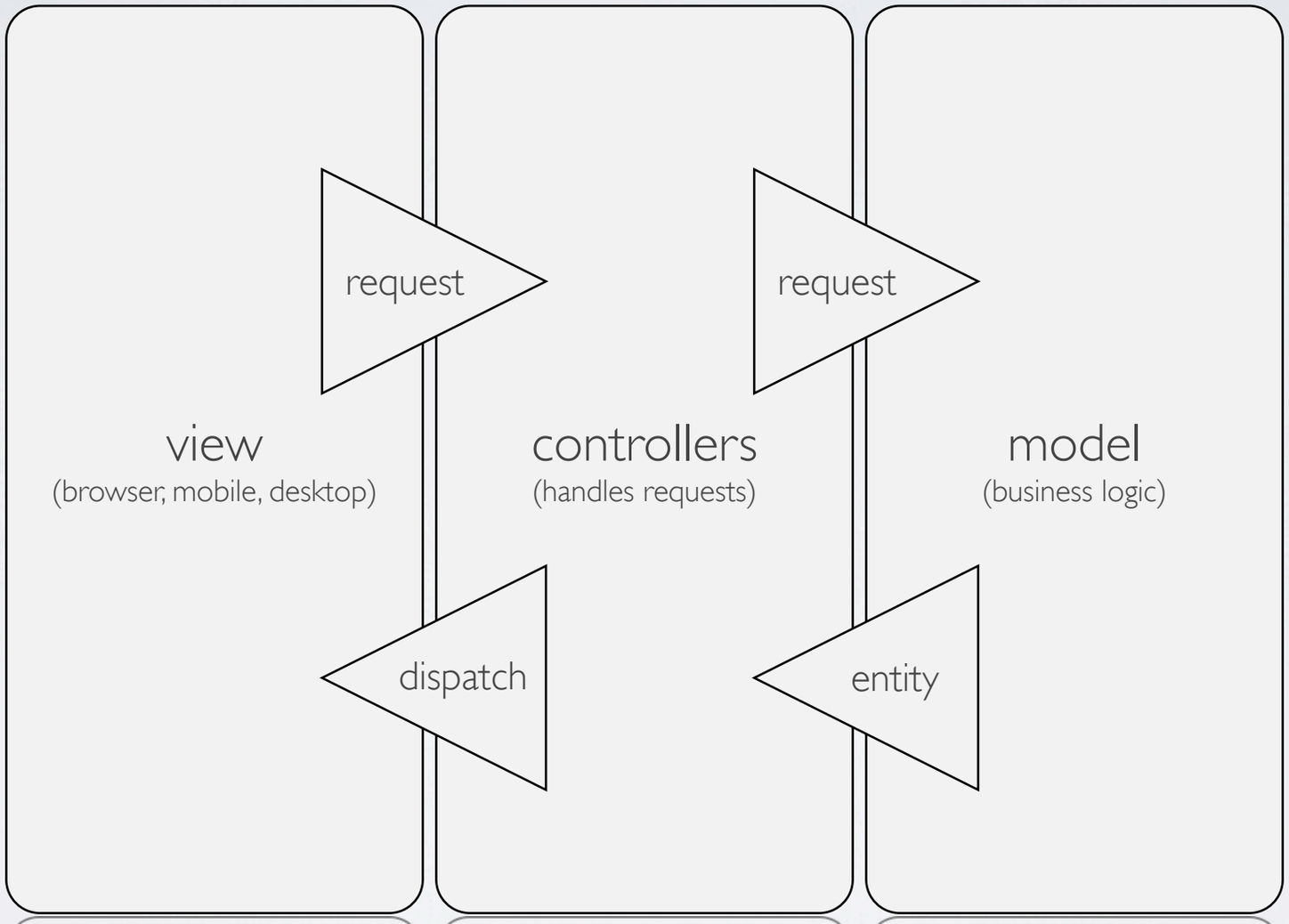
*Great for applications



patrones
arquitectura

mvc

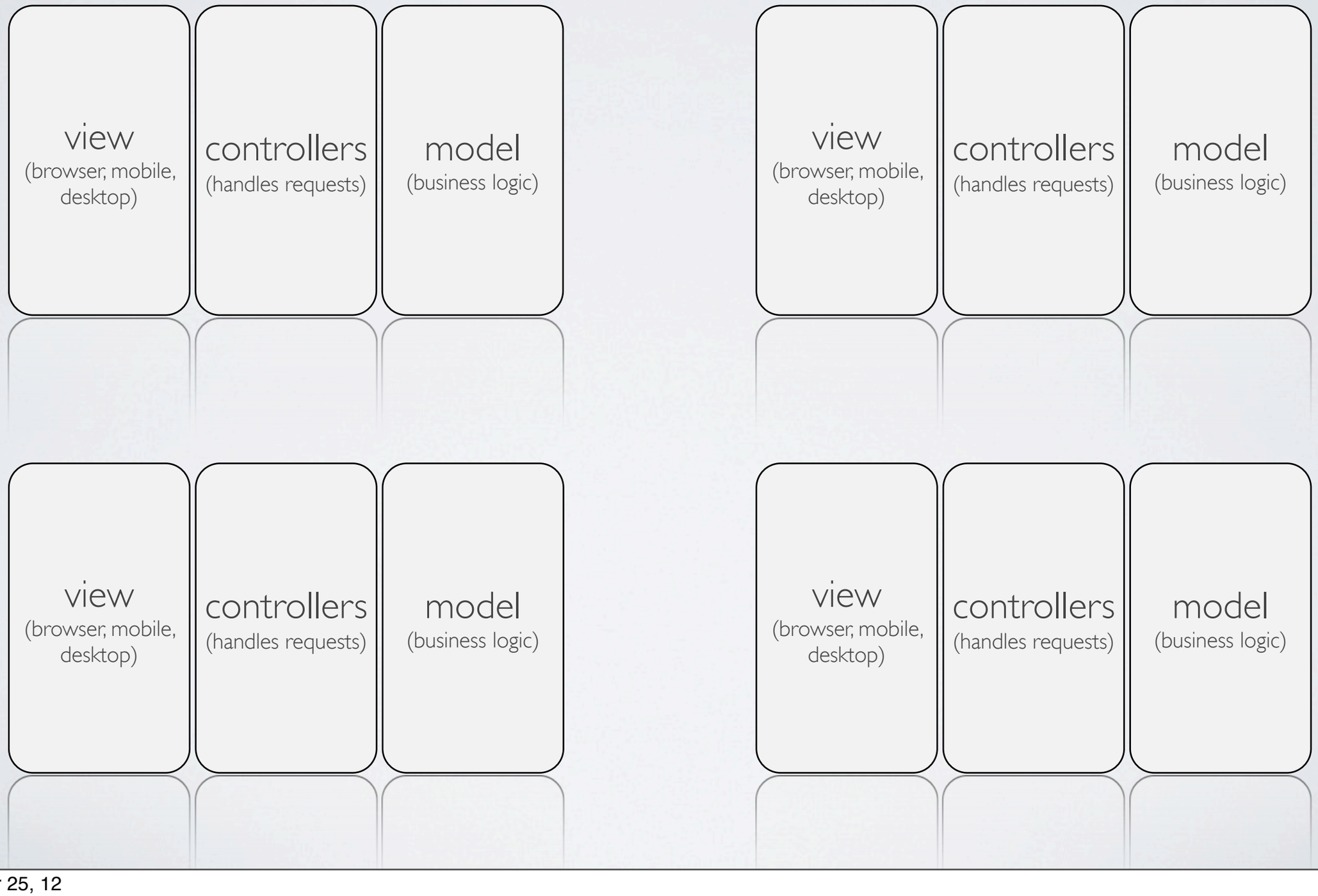
*Great for applications



anti.patrones

arquitectura

mvc / silver bullet mvc

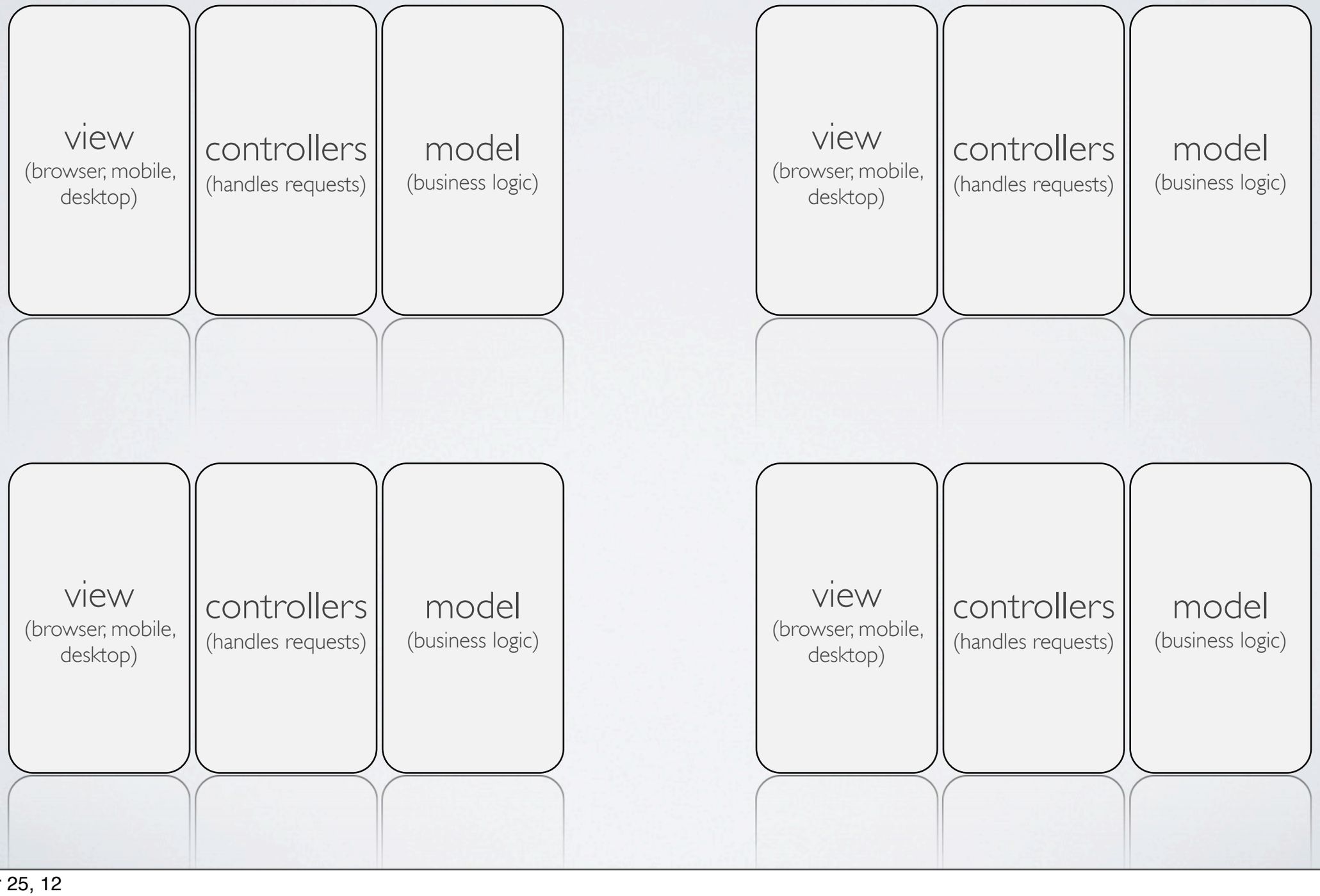


anti.patrones

arquitectura

*So extended most programmer don't know any other alternatives.
Not a good solution for all systems.
e.g. event based systems.

mvc / silver bullet mvc



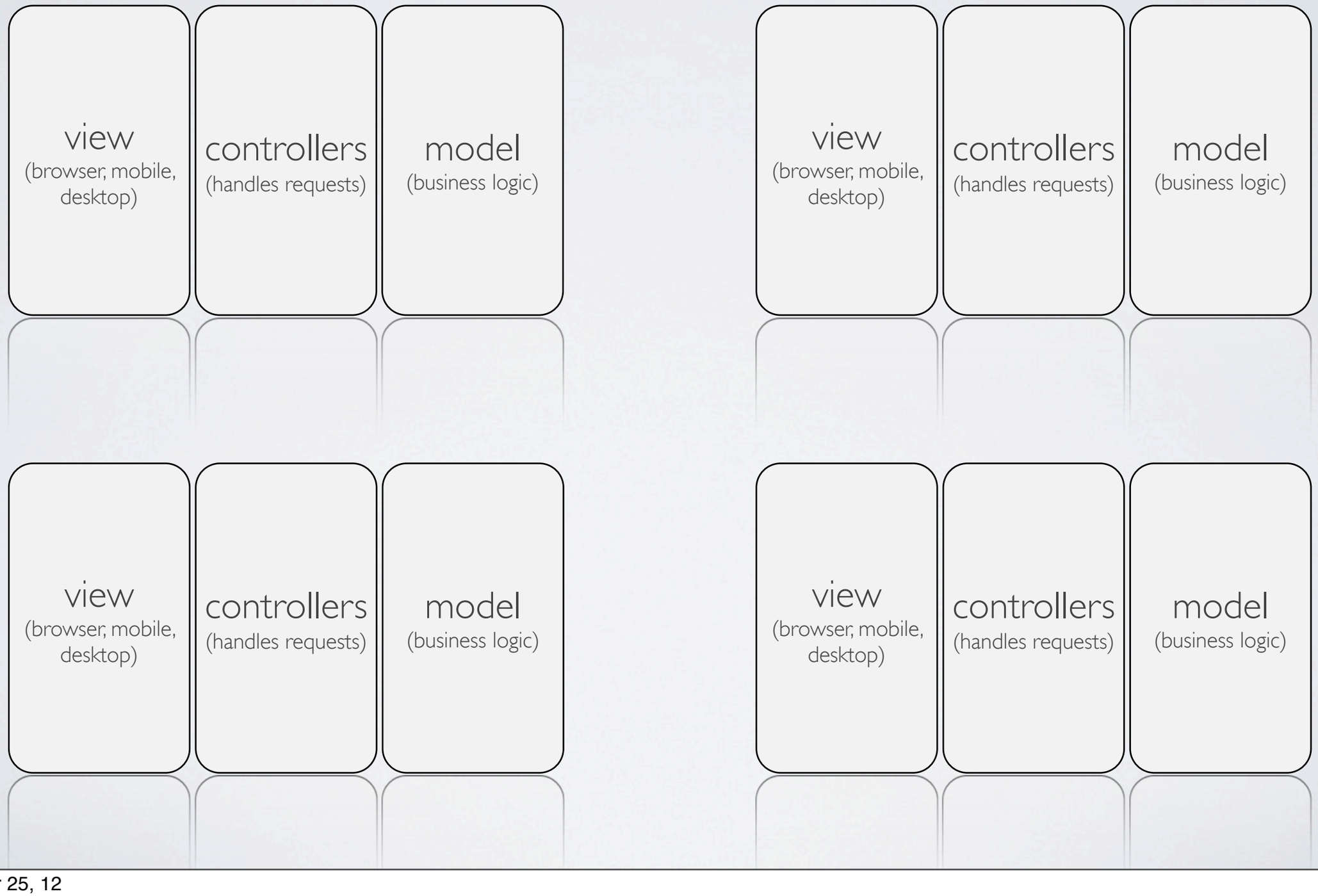
anti.patrones

arquitectura

*So extended most programmer don't know any other alternatives.
Not a good solution for all systems.
e.g. event based systems.

mvc / silver bullet mvc

*apps



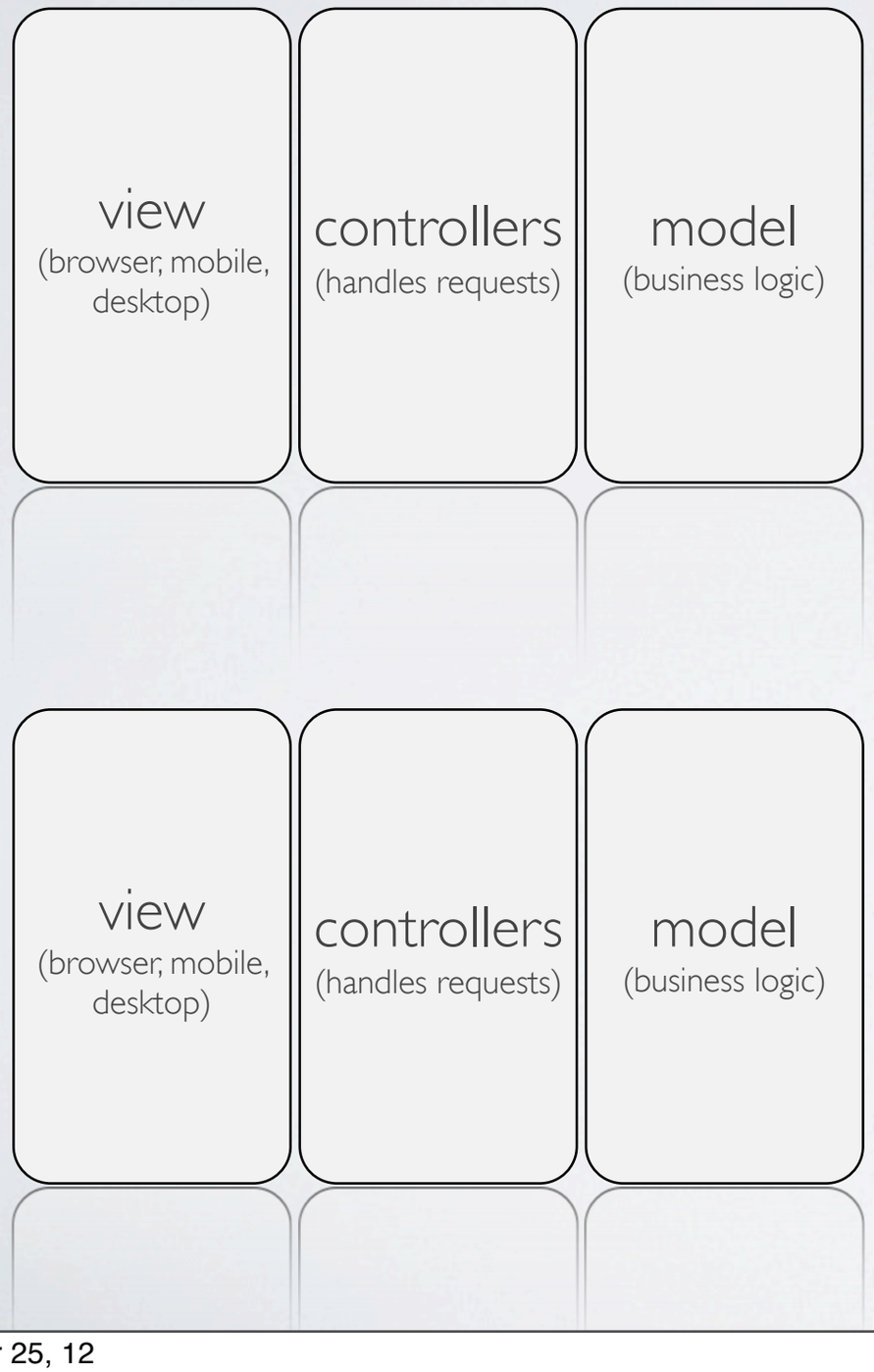
anti.patrones

arquitectura

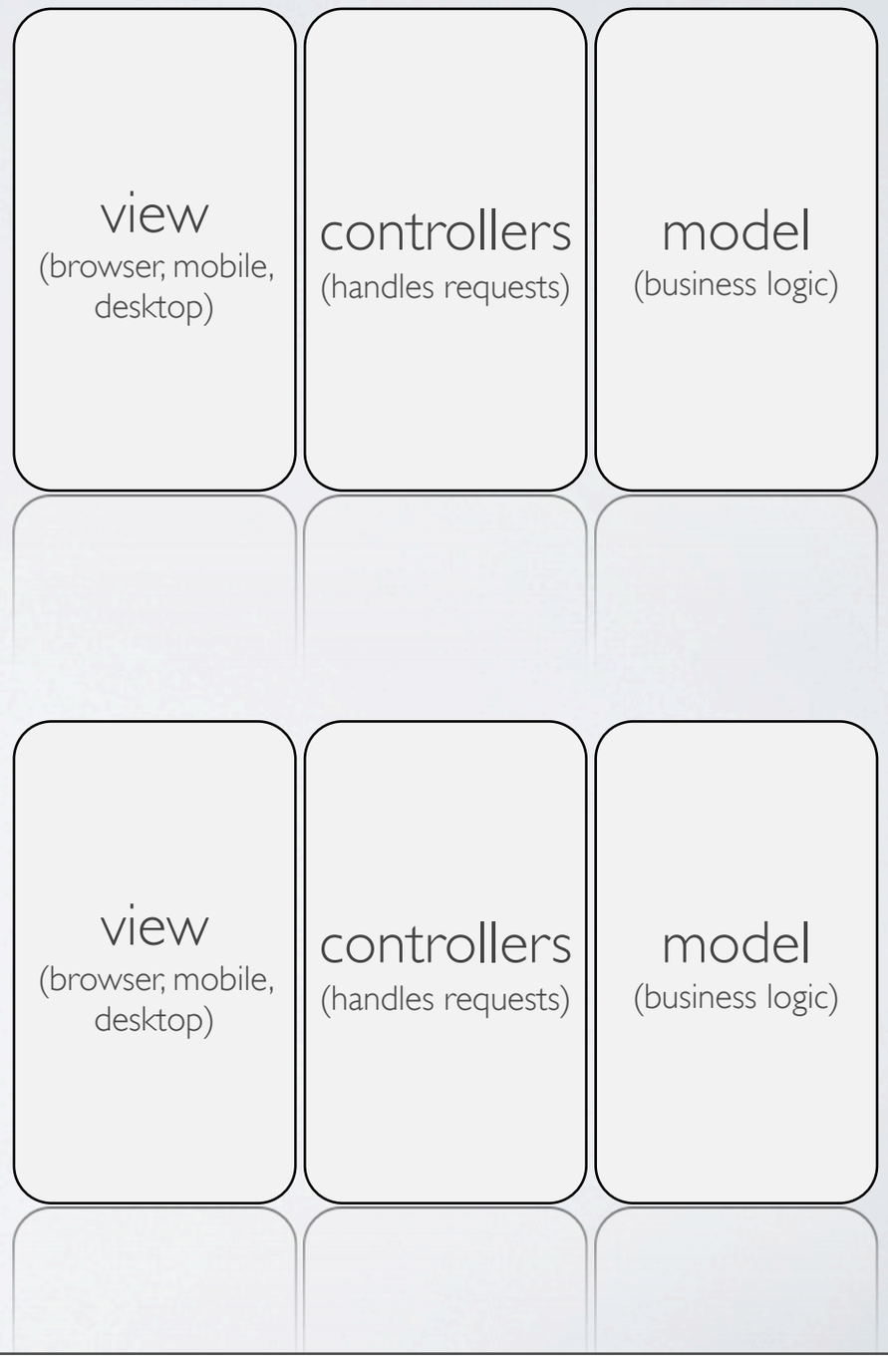
*So extended most programmer don't know any other alternatives.
Not a good solution for all systems.
e.g. event based systems.

mvc / silver bullet mvc

*apps



*libraries



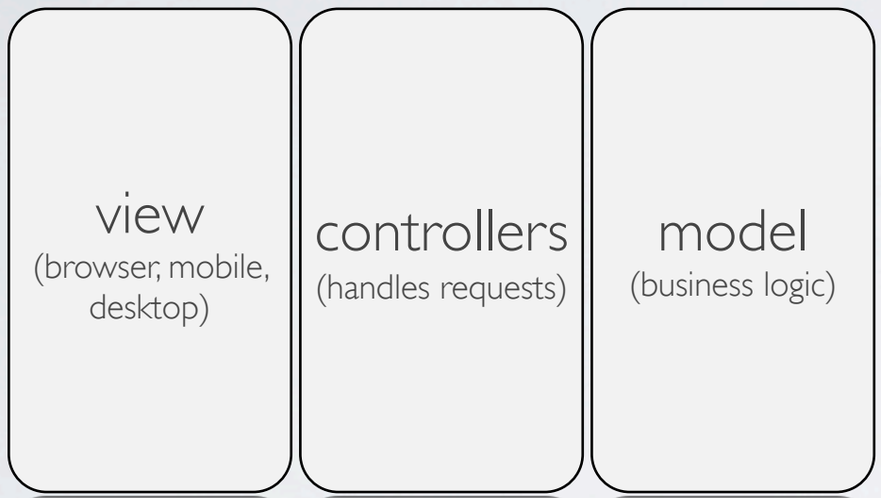
anti.patrones

arquitectura

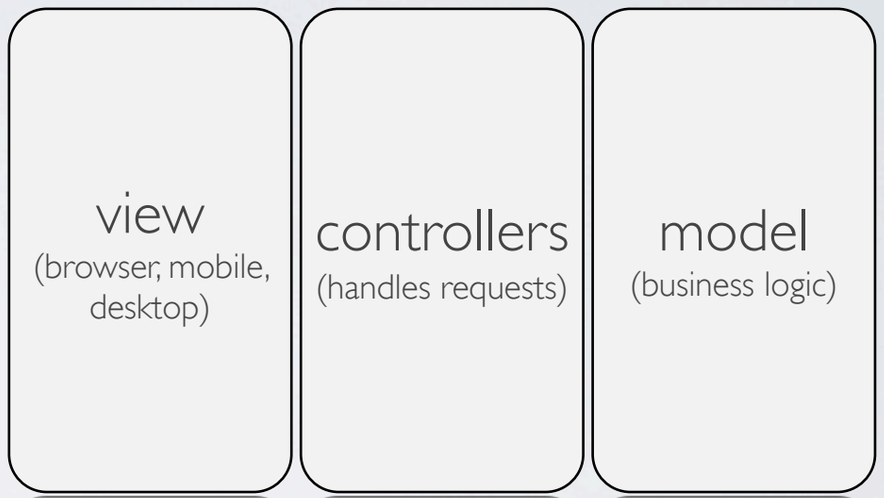
*So extended most programmer don't know any other alternatives.
Not a good solution for all systems.
e.g. event based systems.

mvc / silver bullet mvc

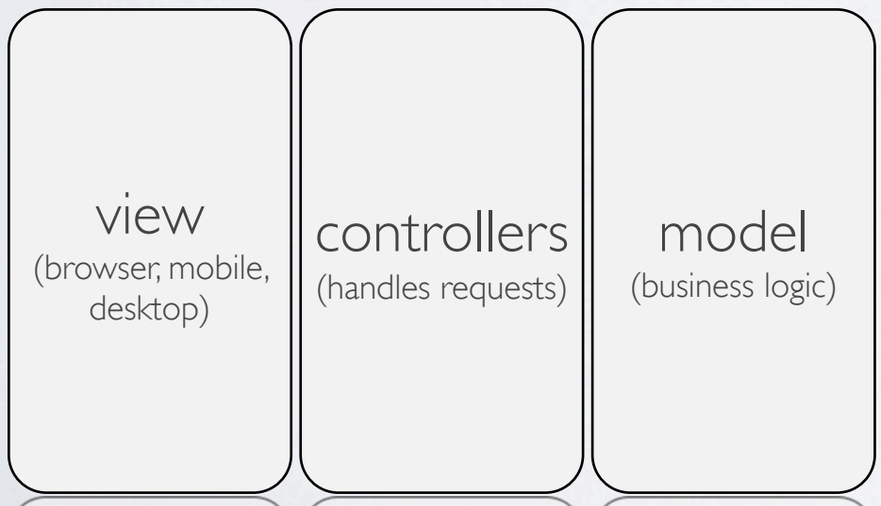
*apps

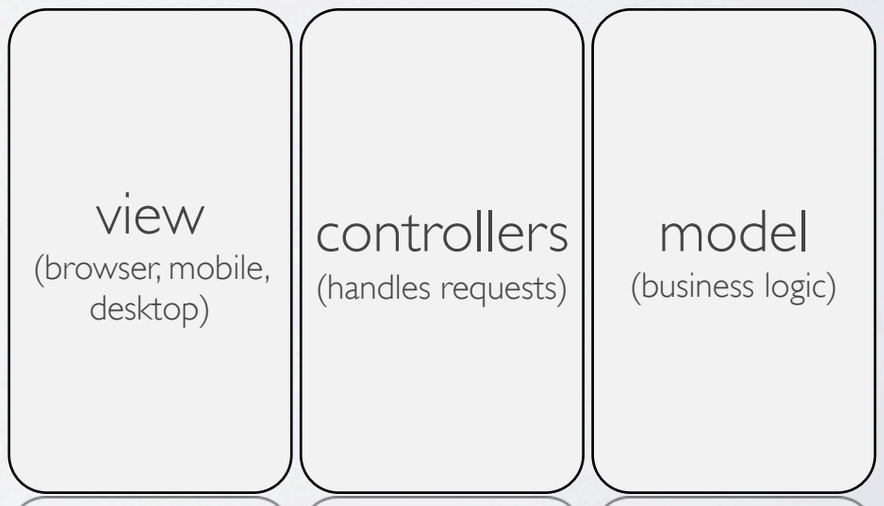


*libraries



*frameworks





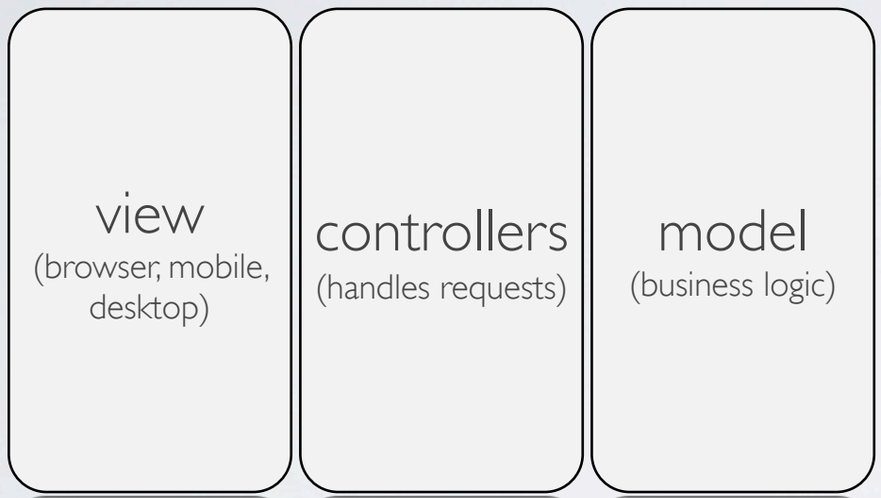
anti.patrones

arquitectura

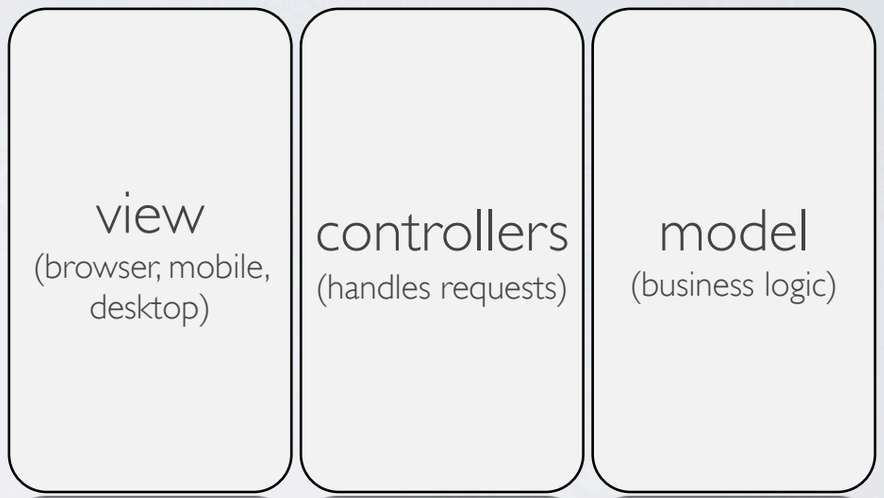
*So extended most programmer don't know any other alternatives.
Not a good solution for all systems.
e.g. event based systems.

mvc / silver bullet mvc

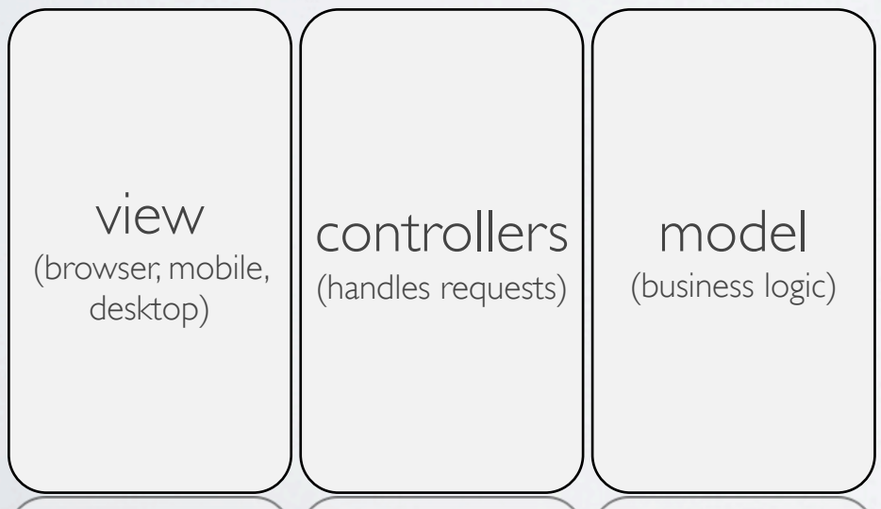
*apps



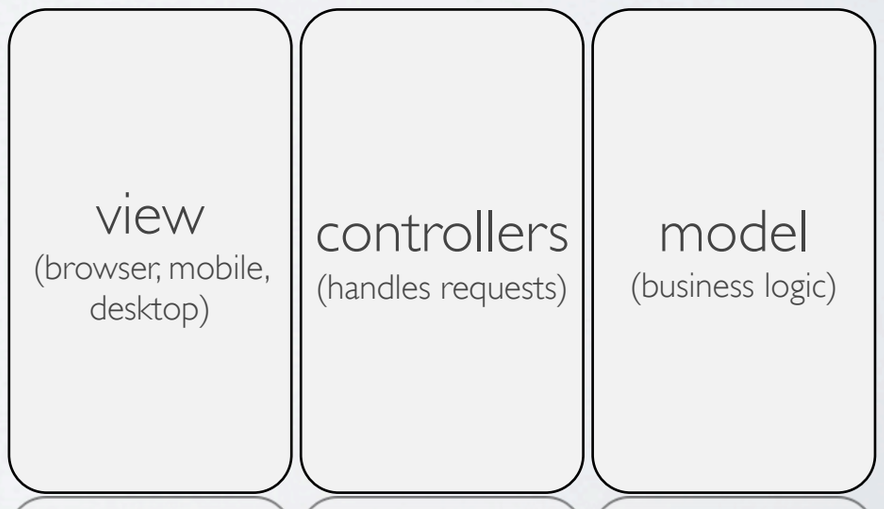
*libraries



*frameworks



*anything



anti.patrones

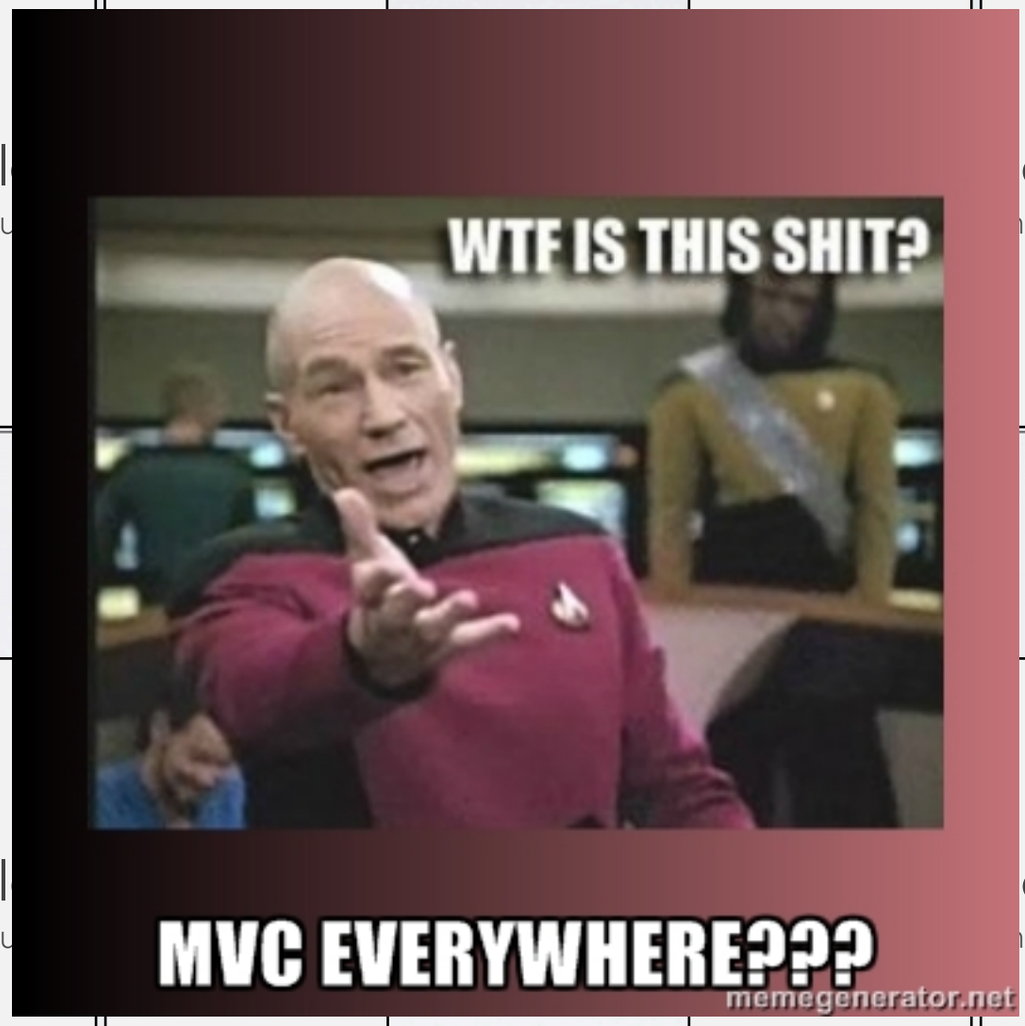
arquitectura

*So extended most programmer don't know any other alternatives.
Not a good solution for all systems.
e.g. event based systems.

mvc / silver bullet mvc

*apps

view (browser, mobile, desktop)	controll (handles requ		*libraries	ontrollers handles requests)	model (business logic)



*frameworks

view (browser, mobile, desktop)	controll (handles requ			ontrollers handles requests)	model (business logic)

patrones
diseño
singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

patrones
diseño
singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

patrones
diseño
singleton

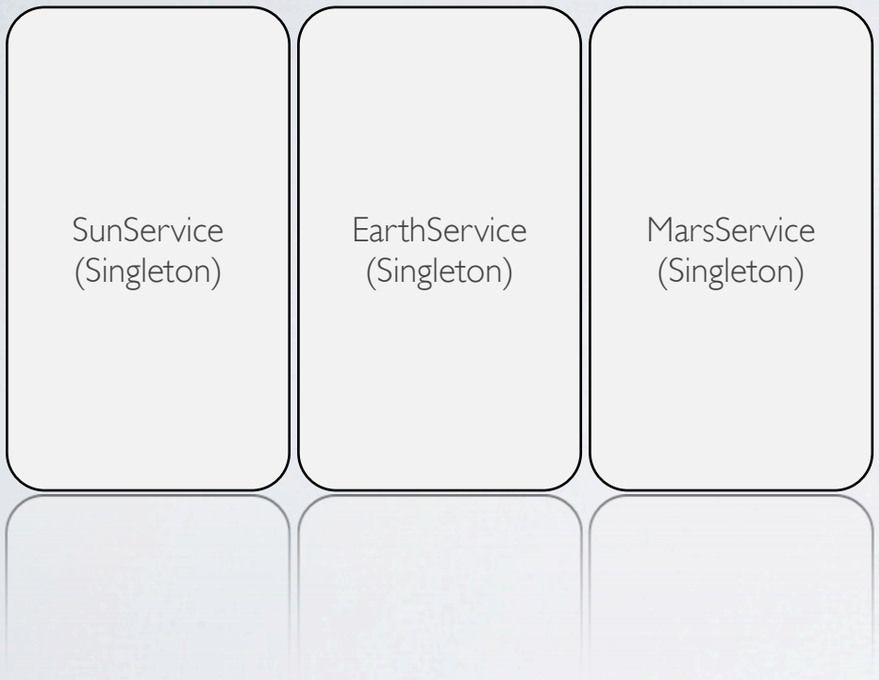
*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

patrones

diseño

singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance



patrones
diseño
singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

SunService
(Singleton)

EarthService
(Singleton)

MarsService
(Singleton)

Fire
(Not a Singleton)

patrones
diseño
singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

SunService
(Singleton)

EarthService
(Singleton)

MarsService
(Singleton)

Fire
(Not a Singleton)

Human
(Not a Singleton)

patrones
diseño
singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

SunService (Singleton)	EarthService (Singleton)	MarsService (Singleton)
Fire (Not a Singleton)	Human (Not a Singleton)	Alien (Not a Singleton)

patrones
diseño
singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

SunService (Singleton)	EarthService (Singleton)	MarsService (Singleton)
Fire (Not a Singleton)	Human (Not a Singleton)	Alien (Not a Singleton)

EarthService.java #

```
1  public class EarthService {
2
3      private final static EarthService instance = new EarthService();
4
5      private EarthService() {
6      }
7
8      public final static EarthService getInstance() {
9          return instance;
10     }
11
12 }
```

patrones
diseño
singleton

*Single instance in memory.
Great for services. Requires manual thread synchronization.
Hard to test unless you use an abstract factory to construct the impl instance

SunService (Singleton)	EarthService (Singleton)	MarsService (Singleton)
Fire (Not a Singleton)	Human (Not a Singleton)	Alien (Not a Singleton)

https://gist.github.com/3928549

EarthService.java #

```
1  public class EarthService {
2
3      private final static EarthService instance = new EarthService();
4
5      private EarthService() {
6      }
7
8      public final static EarthService getInstance() {
9          return instance;
10     }
11
12 }
```

anti.patrones

diseño

singleton/singletonitis

SunService (Singleton)	EarthService (Singleton)	MarsService (Singleton)
Fire (Singleton)	Human (Singleton)	Alien (Singleton)

https://gist.github.com/3928549

EarthService.java #

```
1  public class EarthService {
2
3      private final static EarthService instance = new EarthService();
4
5      private EarthService() {
6      }
7
8      public final static EarthService getInstance() {
9          return instance;
10     }
11
12 }
```

anti.patrones

diseño

singleton/singletonitis

*When devs confuse the factory method and make everything a singleton unknowingly sharing state throughout the app

SunService (Singleton)	EarthService (Singleton)	MarsService (Singleton)
Fire (Singleton)	Human (Singleton)	Alien (Singleton)

https://gist.github.com/3928549

EarthService.java #

```
1  public class EarthService {
2
3      private final static EarthService instance = new EarthService();
4
5      private EarthService() {
6      }
7
8      public final static EarthService getInstance() {
9          return instance;
10     }
11
12 }
```

patrones
diseño
builder

*Helps building complex object out of a series of **optional** parameters

https://github.com/47deg/firebrand#programmatically

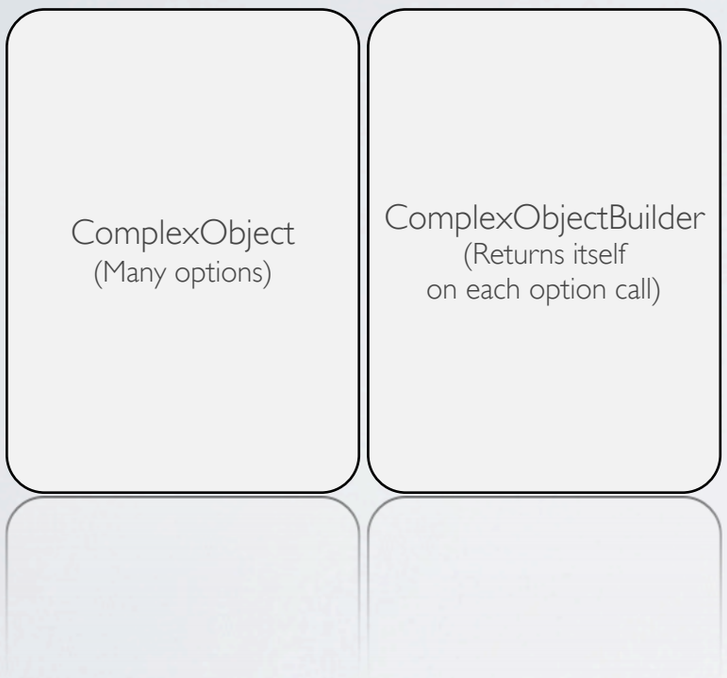
patrones
diseño
builder

*Helps building complex object out of a series of **optional** parameters

https://github.com/47deg/firebrand#programmatically

patrones
diseño
builder

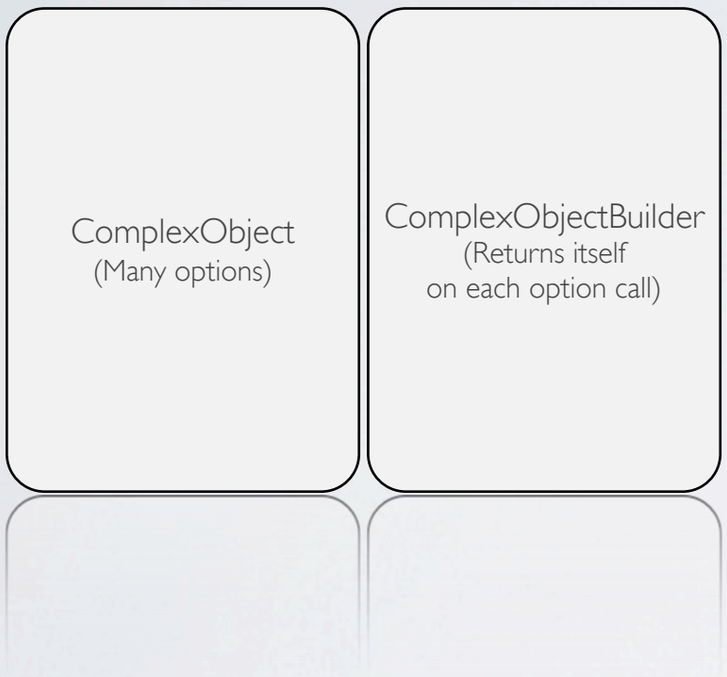
*Helps building complex object out of a series of **optional** parameters



https://github.com/47deg/firebrand#programmatically

patrones
diseño
builder

*Helps building complex object out of a series of **optional** parameters

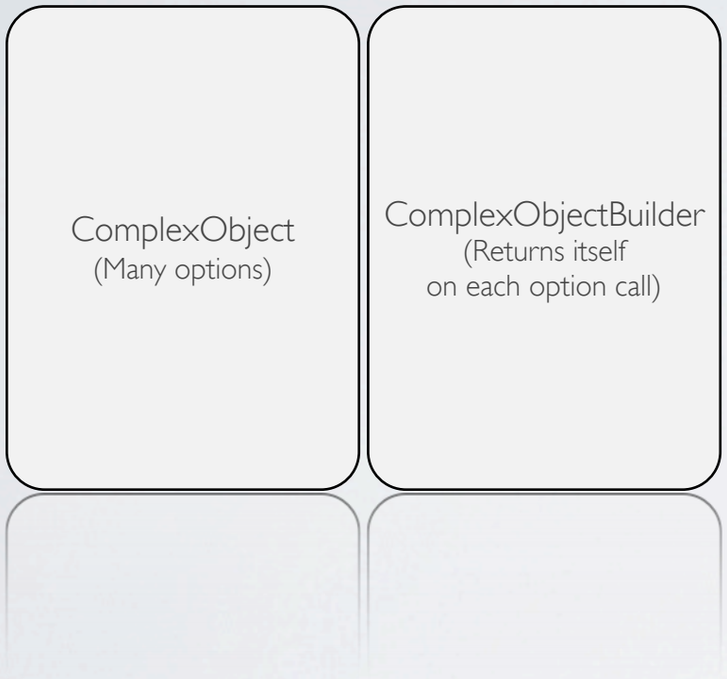


https://github.com/47deg/firebrand#programmatically

```
PersistenceFactory persistenceFactory = new HectorPersistenceFactory.Builder()
                        .defaultConsistencyLevel(true)
                        .clusterName(cluster)
                        .defaultKeySpace(keySpace)
                        .contactNodes(nodes)
                        .thriftPort(port)
                        .autoDiscoverHosts(autoDiscoverHosts)
                        .entities(entities)
                        .build();
```

patrones
diseño
builder

*Helps building complex object out of a series of **optional** parameters




```
843     /**
844      * A builder for this factory impl
845      */
846     public static final class Builder {
847
848         private HectorPersistenceFactory delegate;
849
850         public Builder() {
851             delegate = new HectorPersistenceFactory();
852         }
853
854         public Builder autoDiscoverHosts(boolean autoDiscoverHosts) {
855             delegate.setAutoDiscoverHosts(autoDiscoverHosts);
856             return this;
857         }
858
859         public Builder clusterName(String clusterName) {
860             delegate.setClusterName(clusterName);
861             return this;
862         }
863
864         public Builder credentials(Map<String, String> credentials) {
865             delegate.setCredentials(credentials);
866             return this;
867         }
868
```

https://github.com/47deg/firebrand#programmatically

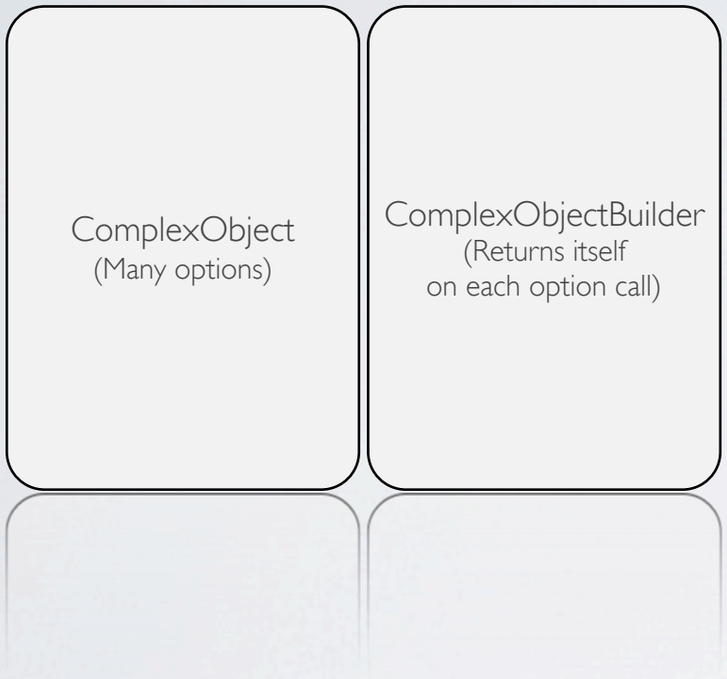
```
PersistenceFactory persistenceFactory = new HectorPersistenceFactory.Builder()
                        .defaultConsistencyLevel(true)
                        .clusterName(cluster)
                        .defaultKeySpace(keySpace)
                        .contactNodes(nodes)
                        .thriftPort(port)
                        .autoDiscoverHosts(autoDiscoverHosts)
                        .entities(entities)
                        .build();
```

anti.patrones

diseño

builder/required

*anti-pattern when some of the params are required. Required params should go on the constructor or factory method




```
843     /**
844      * A builder for this factory impl
845      */
846     public static final class Builder {
847
848         private HectorPersistenceFactory delegate;
849
850         public Builder() {
851             delegate = new HectorPersistenceFactory();
852         }
853
854         public Builder autoDiscoverHosts(boolean autoDiscoverHosts) {
855             delegate.setAutoDiscoverHosts(autoDiscoverHosts);
856             return this;
857         }
858
859         public Builder clusterName(String clusterName) {
860             delegate.setClusterName(clusterName);
861             return this;
862         }
863
864         public Builder credentials(Map<String, String> credentials) {
865             delegate.setCredentials(credentials);
866             return this;
867         }
868
```

https://github.com/47deg/firebrand#programmatically

```
PersistenceFactory persistenceFactory = new HectorPersistenceFactory.Builder()
                        .defaultConsistencyLevel(true)
                        .clusterName(cluster)
                        .defaultKeySpace(keySpace)
                        .contactNodes(nodes)
                        .thriftPort(port)
                        .autoDiscoverHosts(autoDiscoverHosts)
                        .entities(entities)
                        .build();
```

patrones

diseño

factory method

*Helps controlling instance creation by providing an static method that returns an instance

getInstance()

EarthService.java #

```java
public class EarthService {

    private final static EarthService instance = new EarthService();

    private EarthService() {
    }

    public final static EarthService getInstance() {
        return instance;
    }

}
```

anti.patrones

diseño

factory method/reduce visibility

*Anti-pattern when it becomes just a constructor clone and restrict instance creation for no reason.
Consider abstract factories for most cases.

getInstance()

EarthService.java #

```
1  public class EarthService {
2
3      private final static EarthService instance = new EarthService();
4
5      private EarthService() {
6      }
7
8      public final static EarthService getInstance() {
9          return instance;
10     }
11
12 }
```

patrones

diseño

abstract factory

*Helps controlling instance creation by providing a service that give you instances of implementations without directly exposing implementations.
It allows to replace dependencies based on implementations via configuration or at runtime without coupling your code to specific implementations.

https://gist.github.com/3930813

Application.java #

```java
interface PersistenceService {
    void save();
}

class DatabasePersistenceService implements PersistenceService {

    public void save() {
        // save to database
    }

}

class MemoryPersistenceService implements PersistenceService {

    public void save() {
        // save to memory
    }

}


class ServiceFactory {

    private static PersistenceService persistenceService;

    public static PersistenceService getPersistenceService() {
        return persistenceService;
    }

    public static void setPersistenceService(PersistenceService persistenceService) {
        ServiceFactory.persistenceService = persistenceService;
    }
}

public class Application {

    public static void main(String[] args) {

        ServiceFactory.setPersistenceService(new DatabasePersistenceService());
        PersistenceService persistenceService = ServiceFactory.getPersistenceService();
        persistenceService.save();

    }

}
```

anti.patrones

diseño

abstract factory

*The most common anti-pattern is to not specify interfaces so that your code ends up depending on implementation. Another anti-pattern is to have interfaces that are too generic such as in a CRUD api example where all the business logic is delegated to factory clients.

https://gist.github.com/3930813

Application.java #

```java
interface PersistenceService {
    void save();
}

class DatabasePersistenceService implements PersistenceService {

    public void save() {
        // save to database
    }

}

class MemoryPersistenceService implements PersistenceService {

    public void save() {
        // save to memory
    }

}


class ServiceFactory {

    private static PersistenceService persistenceService;

    public static PersistenceService getPersistenceService() {
        return persistenceService;
    }

    public static void setPersistenceService(PersistenceService persistenceService) {
        ServiceFactory.persistenceService = persistenceService;
    }
}

public class Application {

    public static void main(String[] args) {

        ServiceFactory.setPersistenceService(new DatabasePersistenceService());
        PersistenceService persistenceService = ServiceFactory.getPersistenceService();
        persistenceService.save();

    }

}
```

patrones
diseño
façade

*Simplifies and unifies access to a set of more complex functionality

https://gist.github.com/3930928

Application.java #

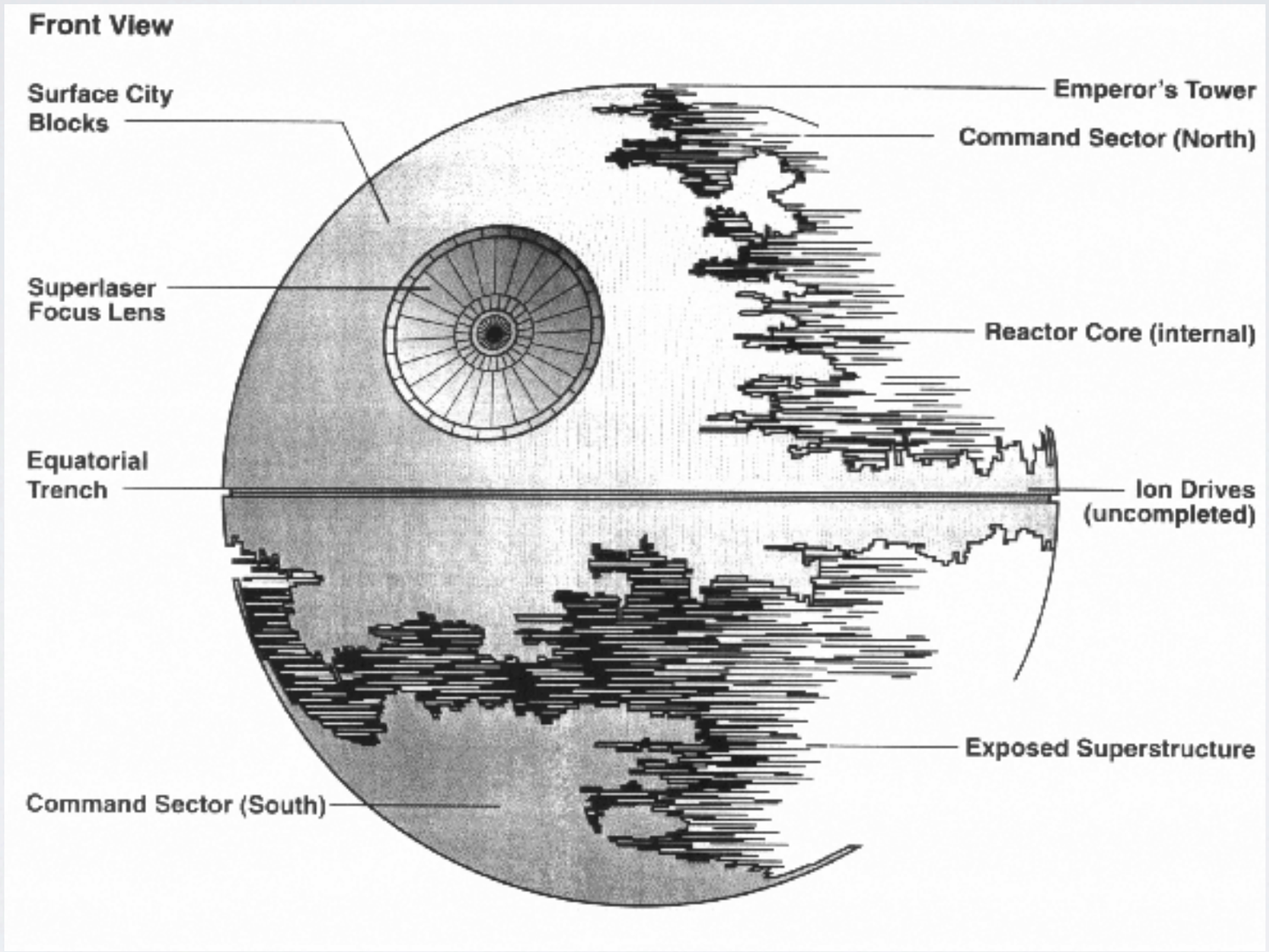
```
1  interface SolidObject {
2      void applyGravity();
3      void colide(SolidObject other);
4  }
5
6  interface MovingObject {
7      void run();
8      void stop();
9  }
10
11 ....
12
13 class Car {
14
15     SolidObject solidObject;
16
17     MovingObject movingObject;
18
19     void go() {
20         solidObject.applyGravity();
21         movingObject.run();
22     }
23
24     void crash(Car other) {
25         movingObject.stop();
26         solidObject.colide(other.solidObject);
27         other.crash(this);
28     }
29
30 }
```

anti.patrones

diseño

façade/death star

* A common anti-pattern is the death start or god object. Where the facade is simplified to the point that its implementation includes too much functionality in a generic way that makes updates to the code or modifications a nightmare due to its complexity



patrones
diseño

proxy

*Proxies calls to class methods allowing interception and modification. Read up on Spring AOP and AspectJ for more powerful actions

PersistenceServiceProxy.java #

```
1
2  interface PersistenceService {
3      Object save(Object obj);
4  }
5
6  class DatabasePersistenceService implements PersistenceService {
7      @Override
8      public Object save(Object obj) {
9          //save to the database
10         return null;
11     }
12 }
13
14 public class PersistenceServiceProxy implements InvocationHandler {
15
16     private Object obj;
17
18     public static Object newInstance(Object obj) {
19         return java.lang.reflect.Proxy.newProxyInstance(
20                 obj.getClass().getClassLoader(),
21                 obj.getClass().getInterfaces(),
22                 new PersistenceServiceProxy(obj));
23     }
24
25     private PersistenceServiceProxy(Object obj) {
26         this.obj = obj;
27     }
28
29     public Object invoke(Object proxy, Method m, Object[] args)
30             throws Throwable {
31         Object result;
32         try {
33             System.out.println("before method " + m.getName());
34             result = m.invoke(obj, args);
35         } catch (InvocationTargetException e) {
36             throw e.getTargetException();
37         } catch (Exception e) {
38             throw new RuntimeException("unexpected invocation exception: " +
39                     e.getMessage());
40         } finally {
41             System.out.println("after method " + m.getName());
42         }
43         return result;
44     }
45
46     public static void main(String... args) {
47         PersistenceService proxy = (PersistenceService) PersistenceServiceProxy.newInstance(new DatabasePersistenceService());
48         proxy.save(new Object());
49     }
50 }
```

anti.patrones

diseño

proxy / hidden code

*Proxied calls are hard to debug because unless you step in the debugger and look at the call stacks is not apparent who is intercepting your code

otros.patrones

- Patrones y Antipatrones
  - Arquitectura
    - **Capas / YAFL**
    - **MVC / MVC**
    - SOA / CRUD
  - Diseño
    - Creación
      - Prototipo
      - **Singleton / Singletonitis**
      - **Builder / Utilizar como constructor**
      - **Método de Factoria / Parametros opcionales**
      - **Factoría abstracta / Falta de interfaces**
    - Estructura
      - Adaptador
      - Puente
      - Compuestos
      - Decorador
      - **Façade / Estrella de muerte : objecto dios**
      - **Proxy**
      - Modulos
    - Comportamiento
      - Cadena de responsabilidad
      - Comando
      - Mediador
      - Memento
      - Observador
      - Estado
      - Estrategia
      - Template
      - Visitante
  - Dialectos
    - Especificos a cada lenguage
      - Java Generics
      - JEE / JEE
      - Metadata / Anotaciones